AP Computer Science A
1999 Sample Student Responses

These materials were produced by Educational Testing Service (ETS), which develops and administers the examinations of the Advanced Placement Program for the College Board. The College Board and Educational Testing Service (ETS) are dedicated to the principle of equal opportunity, and their programs, services, and employment policies are guided by that principle.

The College Board is a national nonprofit membership association dedicated to preparing, inspiring, and connecting students to college and opportunity. Founded in 1900, the association is composed of more than 3,900 schools, colleges, universities, and other educational organizations. Each year, the College Board serves over three million students and their parents, 22,000 high schools, and 3,500 colleges, through major programs and services in college admission, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT®, the PSAT/NMSQT™, the Advanced Placement Program® (AP®), and Pacesetter®. The College Board is committed to the principles of equity and excellence, and that commitment is embodied in all of its programs, services, activities, and concerns.

C

2. This question involves reasoning about the code from the Large Integer case study. A copy of the code is provided as part of this examination.

(a) Write a new `BigInt` member function `Div2`, as started below. `Div2` should change the value of the `BigInt` to be the original value divided by 2 (integer division). Assume the `BigInt` is greater than or equal to `0`. One algorithm for implementing `Div2` is:

```
1. Initialize a variable carryDown to 0.

2. For each digit, d, starting with the most significant digit,

    2.1 replace that digit with (d / 2) + carryDown

    2.2 let carryDown be (d % 2) * 5

3. Normalize the result.
```

Complete member function `Div2` below.

```
void BigInt::Div2()
// precondition:  BigInt ≥ 0
{
    int carryDown = 0, d;
    for(int i=myNumDigits-1; i>=0; i--)
    {
        d= GetDigit(i);
        ChangeDigit(i, (d/2)+carryDown);
        carryDown = (d%2) * 5;
    }
    Normalize();
}
```

(b) Write a definition to overload the / operator, as started below. Assume that dividend and divisor are both positive values of type BigInt.

For example, assume that bigNum1 and bigNum2 are positive values of type BigInt:

bigNum1	bigNum2	bigNum1 / bigNum2
18	9	2
17	2	8
8714	2178	4
9990	999	10

There are many ways to implement division; however, you must use a binary search algorithm to find the quotient of dividend divided by divisor in this problem. You will receive no credit on this part if you do not use a binary search algorithm.

One possible algorithm for implementing division using binary search is as follows:

Let low and high represent a range in which the quotient is found.

Initialize low to 0 and high to dividend.

For each iteration, compute mid = (low + high + 1), divide mid by 2, and compare mid * divisor with dividend to maintain the invariant that low ≤ quotient and high ≥ quotient.

When low == high, the quotient has been found.

In writing function operator/ you may call function Div2 specified in part (a). Assume that Div2 works as specified, regardless of what you wrote in part (a). You will receive NO credit on this part if you do not use a binary search algorithm.

Complete operator/ below. Assume that operator/ is called only with parameters that satisfy its precondition.

```
BigInt operator/ (const BigInt & dividend, const BigInt & divisor)
// precondition:  dividend > 0, divisor > 0
{
   BigInt  low(0),  high(dividend),  mid;
   while(low != high)
   {
      mid = (low + high + 1);
      mid.Div2();
      if(mid * divisor > dividend)
         high = mid - 1;
      else
         low = mid;
   }
   return  low;
}
```

G

2. This question involves reasoning about the code from the Large Integer case study. A copy of the code is provided as part of this examination.

   (a) Write a new `BigInt` member function `Div2`, as started below. `Div2` should change the value of the `BigInt` to be the original value divided by 2 (integer division). Assume the `BigInt` is greater than or equal to `0`. One algorithm for implementing `Div2` is:

   ```
   1. Initialize a variable carryDown to 0.

   2. For each digit, d, starting with the most significant digit,

       2.1 replace that digit with (d / 2) + carryDown

       2.2 let carryDown be (d % 2) * 5

   3. Normalize the result.
   ```

   Complete member function `Div2` below.

```
void BigInt::Div2()
// precondition:  BigInt ≥ 0
```

~~{~~
~~int CarryDown(0);~~
~~int d;~~
~~for (int i=0; i < NumDigits();i++) {~~
~~d= GetDigit(i);~~
~~ChangeDigit(i, (d/2) + CarryDown);~~
~~CarryDown = (d%2) * 5;~~
~~}~~
~~Normalize();~~
~~}~~

```
{
  int carrydown (0);
  int d;
  for (int i=0; i < NumDigits(); i++) {
     d= GetDigit(i);
     ChangeDigit(i, (d / 2)+ carrydown);
     carrydown = (d % 2) * 5;
  }
  Normalize();
}
```

GO ON TO THE NEXT PAGE

(b) Write a definition to overload the / operator, as started below. Assume that dividend and divisor are both positive values of type BigInt.

For example, assume that bigNum1 and bigNum2 are positive values of type BigInt:

bigNum1	bigNum2	bigNum1 / bigNum2
18	9	2
17	2	8
8714	2178	4
9990	999	10

There are many ways to implement division; however, you must use a binary search algorithm to find the quotient of dividend divided by divisor in this problem. You will receive no credit on this part if you do not use a binary search algorithm.

One possible algorithm for implementing division using binary search is as follows:

Let low and high represent a range in which the quotient is found.

Initialize low to 0 and high to dividend.

For each iteration, compute mid = (low + high + 1), divide mid by 2, and compare

mid * divisor with dividend to maintain the invariant that low ≤ quotient and

high ≥ quotient.

When low == high, the quotient has been found.

In writing function operator/ you may call function Div2 specified in part (a). Assume that Div2 works as specified, regardless of what you wrote in part (a). You will receive NO credit on this part if you do not use a binary search algorithm.

Complete operator/ below. Assume that operator/ is called only with parameters that satisfy its precondition.

```
BigInt operator/ (const BigInt & dividend, const BigInt & divisor)
// precondition:  dividend > 0, divisor > 0
{
  BigInt low(0); high(dividend);
  BigInt mid, quotient(1);
   while ( low != high) {
      mid = (low + high +1);
      mid. Div2();
       quotient = mid * divisor;
       if  (dividend > quotient)  low = mid;
       if  (dividend < quotient)  high = mid;
   }
   return low;
  }
```

GO ON TO THE NEXT PAGE

I

2. This question involves reasoning about the code from the Large Integer case study. A copy of the code is provided as part of this examination.

   (a) Write a new `BigInt` member function `Div2`, as started below. `Div2` should change the value of the `BigInt` to be the original value divided by 2 (integer division). Assume the `BigInt` is greater than or equal to `0`. One algorithm for implementing `Div2` is:

```
1. Initialize a variable carryDown to 0.

2. For each digit, d, starting with the most significant digit,

    2.1 replace that digit with (d / 2) + carryDown

    2.2 let carryDown be (d % 2) * 5

3. Normalize the result.
```

Complete member function `Div2` below.

```
void BigInt::Div2()
// precondition:  BigInt ≥ 0
  {
     int CarryDown == 0;
    for (int i = 0; i < numDigits ;i++)
        {
            ChangeDigit( i, (d / 2) + CarryDown)
            CarryDown = (d%2) * 5;
        }
    Normalize();
  }
```

GO ON TO THE NEXT PAGE

(b) Write a definition to overload the / operator, as started below. Assume that dividend and divisor are both positive values of type BigInt.

For example, assume that bigNum1 and bigNum2 are positive values of type BigInt:

bigNum1	bigNum2	bigNum1 / bigNum2
18	9	2
17	2	8
8714	2178	4
9990	999	10

There are many ways to implement division; however, you must use a binary search algorithm to find the quotient of dividend divided by divisor in this problem. You will receive no credit on this part if you do not use a binary search algorithm.

One possible algorithm for implementing division using binary search is as follows:

Let low and high represent a range in which the quotient is found.

Initialize low to 0 and high to dividend.

For each iteration, compute mid = (low + high + 1), divide mid by 2, and compare

mid * divisor with dividend to maintain the invariant that low ≤ quotient and

high ≥ quotient.

When low == high, the quotient has been found.

In writing function operator/ you may call function Div2 specified in part (a). Assume that Div2 works as specified, regardless of what you wrote in part (a). You will receive NO credit on this part if you do not use a binary search algorithm.

Complete operator/ below. Assume that operator/ is called only with parameters that satisfy its precondition.

```
BigInt operator/ (const BigInt & dividend, const BigInt & divisor)
// precondition:  dividend > 0, divisor > 0
{
  ~~mid = low + high~~
   low = 0;
   high = dividend;
  for(;;)
  while (low ≤ quotient && high ≥ quotient)
     {
        mid=(low + high + 1)
        mid. Div2();
     }
   if (low == high)
     { break;
     }
   BigInt quot (mid);
   return quot;
}
```

GO ON TO THE NEXT PAGE